Express Mail No.: EL485649951US
Date of Deposit: February 9 , 2000

Atty Docket No.: 00P7458US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

This is a U.S. Patent Application for:

Title: **SYSTEM AND METHOD FOR REPORTING THE ADDITION AND DELETION OF TAPI LINE AND PHONE DEVICES IN ABSENCE OF SUCH NOTIFICATION FROM A PBX**

Inventor #1: Mark Bernard Hettish
Address: 112 Kingussie Ct., Cary, NC, 27511
Citizenship: USA

SYSTEM AND METHOD FOR REPORTING THE ADDITION AND DELETION OF TAPI LINE AND PHONE DEVICES IN ABSENCE OF SUCH NOTIFICATION FROM A PBX

RESERVATION OF COPYRIGHT

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention relates to communications systems and, in particular, to a communication system employing a private branch exchange (PBX) and a TAPI interface.

DESCRIPTION OF THE RELATED ART

The Telephony Application Programming Interface (TAPI) is a high level programming interface for Windows™ which supports many types of telephony applications associated with conventional analog public telephone lines, PBX phone lines, ISDN phone lines, and the like. Thus, TAPI allows a communication application to support numerous telephony operations through a variety of mediums by making a function call to TAPI which will drive the hardware (fax/modem card, DSP card, network switch, and the like) coupled thereto.

The TAPI architecture 100 is illustrated in FIG. 1. As shown, the TAPI architecture 100 includes a TAPI implementation 104 interfaced to telephony application programs 102. TAPI 104 provides a connection to a TAPI service provider, such as a TAPI server 106, which then interfaces to hardware such as voices cards 108a, H.323 interfaces 108b, or PBX's 108c.

The TAPI specification requires that device configuration information

be available at the startup of the TAPI service provider. If unsolicited addition and deletion events from a telephony device hardware, such as a PBX, are not supported (i.e., the hardware does not automatically inform the TAPI service provider of updates), the TAPI service provider can only do configuration change updates at start up. The TAPI service provider does so by requesting the static device configuration from the PBX, building the initial internal database and reporting all the devices to the TAPI. If changes are made on the PBX, the telephony server system administrator would be required to shut down the TAPI service provider and restart it again. This has the disadvantage of disrupting service to all users of the TAPI service provider. Moreover, in such a system, there is no way to shut down a TAPI service provider if it is in use. It will be shut down only if every TAPI application using it shuts down. Since several thousand clients can be supported, this can provide severe disadvantages in administration.

SUMMARY OF THE INVENTION

These and other drawbacks in the prior art are overcome in large part by a system and method for client configuration in a TAPI environment according to the present invention. Briefly, the present invention provides a method whereby a TAPI service provider may request configuration information from hardware such as a PBX. Configuration information received from the hardware is stored in a list and compared to the existing configuration list. Two new lists are then generated: an added list and a deleted list. All devices in the deleted list are deleted from the internal database and deleted messages are sent to the TAPI service provider. All new devices are temporarily added to the internal database and addition messages are sent to the TAPI service provider. The new device configuration list is then saved, and the added devices are marked as permanent.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the invention is obtained when the following

detailed description is considered in conjunction with the following drawings in which:

FIG. 1 is a diagram representative of the TAPI architecture;

FIG. 2 is a diagram illustrating a computer system employing a TAPI system according to an implementation of the present invention;

FIG. 3 is a block diagram of the computer system of FIG. 2 according to an implementation of the present invention; and

FIG. 4A and FIG. 4B are flowcharts illustrating operation of an implementation of the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIGS. 2-4 illustrate an improved system and method for a TAPI service provider to request and update configuration information from hardware such as a PBX. Configuration information received from the hardware is stored in a list and compared to the existing configuration list. Two new lists are then generated: an added list and a deleted list. All devices in the deleted list are deleted from the internal database and deleted messages are sent to the TAPI service provider. All new devices are temporarily added to the internal database and addition messages are sent to the TAPI service provider. The new device configuration list is then saved, and the added devices are marked as permanent.

An exemplary TAPI client 202 is shown in FIG. 2. The TAPI client 202 may be embodied as a personal computer, including a system unit 11, a keyboard 12, a mouse 13, and a display 140. Also shown are one or more speakers 150a, 150b, and a microphone 1601. The screen 160 of the display device 14 is used to present a graphical user interface (GUI) and particularly, a TAPI client window 3008. The graphical user interface supported by the operating system allows the user to employ a point and click method of input, i.e., by moving the mouse pointer or cursor (not shown) to an icon representing a data object at a particular location on the screen 160 and pressing one or more of the mouse buttons to perform a user command or selection. The GUI may be any of the Windows GUIs available from

Microsoft Corporation or the Macintosh OS, available from Apple Computer.

FIG. 3 shows a block diagram of the components of the personal computer shown in FIG. 2. The system unit 11 includes a system bus or a plurality of system buses 21 to which various components are coupled and by which communication between the various components is accomplished. The microprocessor 22 is coupled to the system bus 21 and is supported by the read only memory (ROM) 23 and the random access memory (RAM) 24 also connected to the system bus 21. The microprocessor 22 may be embodied as any of a variety of microprocessors, including Intel x86, Pentium or Pentium II or compatible processors.

The ROM 23 contains among other code the basic input output system (BIOS) which controls basic hardware operations such as the interaction of the disk drives and the keyboard. The RAM 24 is the main memory into which the operating system and applications programs are loaded. The memory management chip 25 is connected to the system bus 21 and controls direct memory access operations including passing data between the RAM 24 and hard disk drive 26 and floppy disk drive 27. A CD ROM drive (or DVD or other optical drive) 32 may also be coupled to the system bus 21 and is used to store a large amount of data, such as a multimedia program or a large database.

Also connected to the system bus 21 are various I/O controllers: The keyboard controller 28, the mouse controller 29, the video controller 30, and the audio controller 31. The keyboard controller 28 provides the hardware interface for the keyboard; the mouse controller 29 provides the hardware interface for the mouse 13; the video controller 30 is the hardware interface for the video display 14; and the audio controller 31 is the hardware interface for the speakers 15 and microphone 16. The speakers 150a, b and the microphone 1601 allow for audio communication during telephony operation. In operation, keyboard strokes are detected by the keyboard controller 28 and corresponding signals are transmitted to the microprocessor 22; similarly, mouse movements and button clicks are detected by the mouse controller and provided to the microprocessor 22. Typically, the keyboard controller 28

and the mouse controller 29 assert interrupts at the microprocessor 22. In response, the microprocessor 22 executes a corresponding interrupt routine, as is known. Additionally, an interrupt controller (not shown) may be provided to arbitrate among interrupt requests. An I/O controller or network interface 40 enables communication over a network 46, such as a packet network.

One embodiment of the present invention is as a set of instructions in a code module resident in the RAM 24. Until required by the computer system, the set of instructions may be stored in another computer memory, such as the hard disk 26, on an optical disk for use in the CD ROM drive 32, or a floppy disk for use in the floppy disk drive 27.

As shown in the figure, the operating system 50, the TAPI application 52, the TAPI service provider 53, and one or more configuration table objects 56 are resident in the RAM 24. As is known, the operating system 50 functions to generate a graphical user interface on the display 14. The TAPI application program 52 performs TAPI functionality, including generation of a TAPI client window 3008 (FIG. 2) in the GUI. The TAPI service provider 53 implements an interface to the hardware, as will be described in greater detail below.

Turning now to FIG. 4A and 4B, a flowchart illustrating operation of an implementation of the invention is shown. Upon startup, in a step 402, the TAPI service provider 53 requests static configuration information from the PBX (not shown) or other telephony hardware. Such static configuration information can be static line or static phone device information. In the case of a line device, the configuration information includes the device ID and the number of addresses on the line. In the case of a phone device, the information includes the device ID. Turning back to FIG. 4A, in a step 404, the PBX returns multiple blocks of information of PBX configuration information to the TAPI service provider 53. Next, in a step 406, the TAPI service provider 53 instantiates and builds a static configuration table object (C1).

In normal operation, devices are added or deleted to the PBX without notifications being sent to the TAPI service provider 53. Turning now to FIG.

4B, in a step 408, the administrator may request the TAPI service provider 53 check for PBX configuration changes. In a step 410, the TAPI service provider 53 requests static information from the PBX. In a step 412, multiple blocks of information containing the PBX configuration data arrive at the TAPI service provider 53. In a step 414, the TAPI service provider 53 instantiates and builds a new static configuration table object (C2). This new object becomes the current TAPI configuration object. In a step 416, the TAPI service provider 53 takes the C2 object as an input and produces two new configuration table objects: deleted objects (C3) and added devices (C4).

In a step 418, for each device in C3, the TAPI application 52 is notified that the corresponding line device has been deleted. In a step 420, for each device in C3, the TAPI application 52 is notified that the corresponding phone device has been deleted. In a step 422, for each device in C4, the TAPI application 52 is notified that the corresponding line device has been added. Finally, in a step 424, for each device in C4, the TAPI application 52 is notified that the corresponding phone device has been added.

Further details regarding a specific implementation of a system and method according to the present invention are shown in the attached Appendix. It is noted, however, that the invention is not limited to the specific form set forth herein, but includes such alternatives, modifications and equivalents as can reasonably be included within the spirit and scope of the appended claims.

APPENDIX

```
#include "stdafx.h"
#include <afxmt.h>
#include "tapi.h"
#include "tspi.h"
#include "mu_devls.h"

#include "mdb_opti.h"
#include "mdb_tdat.h"
#include "mdb_siem.h"
#include "mdb_tobj.h"
#include "mdb_mgrs.h"
#include "mdb_mdb.h"

#include "ti_dbapi.h"

#include "ti_build.h"

#include "su.h"
DEFINESOURCEINFO;

#include "su_shdat.h"
#include "cdb_main.h"
#include "sh_pl_ad.h"

#include "su_tcorr.h"

#include "ti_intf.h"
#include "sh_stats.h"

#include "mu_aclm.h"
#include "oss_acl.h"
#include "oss_csta.h"


#include "ugglobal.h"
#include "mp_defs.h"

mapper_db::mapper_db()
:  valid(FALSE), next_device_id(0),
      line_device_id_base(0), phone_device_id_base(0)
{

      mapper_db_sem = CreateSemaphore ( NULL, 1, 1, MAPPER_DB_SEM );
      c_stats_db& stats_db = get_stats_db();


      memset ( &stats_db.admin_stats_ptr-
>eng_statistics.eng_mapper_db_stats,
                        0,
                        sizeof ( ENG_MAPPER_DB_STATS ) );

      if ( mapper_db_sem != NULL )
      {

            valid = TRUE;
      }
}
```

```
mapper_db::~mapper_db()
{

      valid = FALSE;
      CloseHandle ( mapper_db_sem );
}



MAPPER_DB_RC
      mapper_db::do_delete_static_devices (     device_list&
      device_list,

      EVENT_INDICATOR   indicator )
{
      MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;


      cnfg_db& cdb = get_cnfg_db();
      CString extension;
      USHORT      type;
      ULONG plid;
      ULONG ppid;
      DWORD sem_rc;

      while ( device_list.get_next_device (extension, &type, &plid,
&ppid) )
      {
            tapi_static_line *line_p = static_line_mgr.find (
extension );
            tapi_static_phone *phone_p = static_phone_mgr.find (
extension );

            if ( (line_p != NULL) && (phone_p != NULL) )
            {
                  tapi_open_phone * open_phone_p =
                        static_cast<tapi_open_phone *>(phone_p-
>child2_p);

                  if ( open_phone_p != NULL )
                  {

                        sem_rc = WaitForSingleObject ( open_phone_p-
>lock, MAPPER_DB_SEM_WAIT );

                        if ( sem_rc == WAIT_OBJECT_0 )
                        {

                              do_delete_open_phone_device (
open_phone_p );

                              ReleaseSemaphore ( open_phone_p->lock,
1, NULL );
                        }
                        else
                        {
```

```
                                    rc = MAPPER_DB_RC_FAILED;

                                    CString error;
                                    error.Format (
5   _T("mapper_db::do_delete_static_devices(). Could not delete phone.")
    );
                                    su_log_message    (     0,
                                                                    SU_DEBUG,
10          SU_TRACE_NONE,
                                                                    0,
                                                                    L"",
            SOURCENAME,
15                                                                  __LINE__,
                                                                    0,
                                                                    0,
                                                                    NULL,
20          (LPCTSTR)error );
                                }
                        }

25                      tapi_open_line * open_line_p =
                                static_cast<tapi_open_line *>(line_p-
    >child2_p);
                        if ( open_line_p != NULL )
30                      {
                                sem_rc = WaitForSingleObject ( open_line_p-
    >lock, MAPPER_DB_SEM_WAIT );
35                              if ( sem_rc == WAIT_OBJECT_0 )
                                {

                                    do_delete_open_line_device (
    open_line_p );
40
                                    ReleaseSemaphore ( open_line_p->lock,
    1, NULL );
                                }
                                else
45                              {

                                    rc = MAPPER_DB_RC_FAILED;

                                    CString error;
50                                  error.Format (
    _T("mapper_db::do_delete_static_devices(). Could not delete line.")
    );
                                    su_log_message    (     0,
                                                                    SU_DEBUG,
55
            SU_TRACE_NONE,
                                                                    0,
                                                                    L"",
60          SOURCENAME,
                                                                    __LINE__,
                                                                    0,
                                                                    0,
                                                                    NULL,
65
            (LPCTSTR)error );
```

```
                    }
                }

                if ( indicator == EVENT_INDICATOR_SEND_EVENT )
                {
                    c_stats_db& stats_db = get_stats_db();

                    tapi_interface& ti = get_tapi_interface();

                    if ( phone_p != NULL )
                    {
                        SEND_TO_TAPI_RC t_rc =
                            ti.send_phone_remove ( phone_p->devid + phone_device_id_base, 0 );

                        if ( t_rc != SEND_TO_TAPI_RC_GOOD )
                        {
                            CString error;
                            error.Format (
_T("mapper_db::do_delete_static_devices(). Problem sending event: phone remove.") );
                            su_log_message    (    0,
        SU_DEBUG,
        SU_TRACE_NONE,
                                                                    0,
        L"",
        SOURCENAME,
        __LINE__,
        t_rc,
                                                                    0,
        NULL,
        (LPCTSTR)error );
                        }

                        if ( cdb.debug_tracing_on (
DBG_MAPPER_DB_BIT ) )
                        {
                            CString error;
                            error.Format (
_T("mapper_db::do_delete_static_devices(). Sent event: phone remove") );
                            su_log_message    (    0,
        SU_DEBUG,
        SU_TRACE_NONE,
                                                                    0,
        L"",
        SOURCENAME,
```

```
        __LINE__,
        t_rc,
                                                                    0,
        NULL,
        (LPCTSTR)error );
                            }

                            stats_db.admin_stats_ptr->
        eng_statistics.eng_mapper_db_stats.
                                        phones_deleted++;
                        }

                        if ( line_p != NULL )
                        {
                            SEND_TO_TAPI_RC t_rc =
                                ti.send_line_remove ( line_p-
>devid + line_device_id_base, 0 );
                            if ( t_rc != SEND_TO_TAPI_RC_GOOD )
                            {
                                CString error;
                                error.Format (
_T("mapper_db::do_delete_static_devices(). Problem sending event:
line remove.") );
                                su_log_message    (     0,
        SU_DEBUG,
        SU_TRACE_NONE,
                                                                    0,
        L"",
        SOURCENAME,
        __LINE__,
        t_rc,
                                                                    0,
        NULL,
        (LPCTSTR)error );
                            }

                            if ( cdb.debug_tracing_on (
DBG_MAPPER_DB_BIT ) )
                            {
                                CString error;
                                error.Format (
_T("mapper_db::do_delete_static_devices(). Sent event: line remove")
);
                                su_log_message    (     0,
        SU_DEBUG,
```

```
        SU_TRACE_NONE,
                                                                0,
5       L"",
        SOURCENAME,
        __LINE__,
10      t_rc,
                                                                0,
        NULL,
15      (LPCTSTR)error );
                                }

20                              stats_db.admin_stats_ptr->
        eng_statistics.eng_mapper_db_stats.
                                                lines_deleted++;
                            }
25                  }

                    if ( phone_p != NULL )
                    {
30                      if ( cdb.debug_tracing_on (
    DBG_MAPPER_DB_OBJ_BIT ) )
                        {
                            CString error;
35                          error.Format (
    _T("mapper_db::do_delete_static_devices().\n\
        Deleting static phone: Ptr = %lx, Object ID = %ld, Device ID =
    %d, Device base = %d, Quick key = %ld, Extension = %s, Monitor status
    = %d"),
40                                  phone_p,
                                    phone_p->object_id,
                                    phone_p->devid,
                            phone_device_id_base,
                                    phone_p->quick_key,
45                                  phone_p->primary_extension,
                                    phone_p->monitor_on );

                            su_log_message    (     0,
                                                            SU_DEBUG,
50
        SU_TRACE_NONE,
                                                            0,
                                                            L"",
55      SOURCENAME,
                                                            __LINE__,
                                                            0,
                                                            0,
                                                            NULL,
60
        (LPCTSTR)error );
                            }

65                      static_phone_mgr.remove_from_in_use_list (
    phone_p);
```

```
                        phone_p->parent_p = NULL;
                        phone_p->child2_p = NULL;

                        static_phone_mgr.add_to_free_list ( phone_p
);
                    }

                    if ( line_p != NULL )
                    {

                        if ( cdb.debug_tracing_on (
DBG_MAPPER_DB_OBJ_BIT ) )
                        {
                            tapi_static_address * address_p =
                                static_cast<tapi_static_address
*>
                                    (line_p->child1_p );

                            CString error;
                            error.Format (
_T("mapper_db::do_delete_static_devices().\n\
    Deleting static line: Ptr = %lx, Object ID = %ld, Device ID = %d,
Device base = %d, Quick key = %ld, Extension = %s, Monitor status =
%d\n\
      With static address: Ptr = %lx, Object ID = %ld, Device ID =
%d, Quick key = %ld, Extension = %s"),
                                line_p,
                                line_p->object_id,
                                line_p->devid,
                        line_device_id_base,
                                line_p->quick_key,
                                line_p->primary_extension,
                                line_p->monitor_on,
                                address_p,
                                address_p->object_id,
                                address_p->devid,
                                address_p->quick_key,
                                address_p->primary_extension );

                            su_log_message    (     0,
                                                    SU_DEBUG,
      SU_TRACE_NONE,
                                                    0,
                                                    L"",
      SOURCENAME,
                                                    __LINE__,
                                                    0,
                                                    0,
                                                    NULL,
      (LPCTSTR)error );
                        }

                        static_line_mgr.remove_from_in_use_list (
line_p );

                        line_p->parent_p = NULL;
                        line_p->child2_p = NULL;
```

```
                        static_line_mgr.add_to_free_list ( line_p );
                    }
                }
                else
                {

                    rc = MAPPER_DB_RC_FAILED;

                    CString error;
                    error.Format (
_T("mapper_db::do_delete_static_devices(). Bad pointer(s), line =
%lx, phone = %lx."),
                                                line_p, phone_p );
                    su_log_message    (     0,
                                            SU_DEBUG,
                                            SU_TRACE_NONE,
                                            0,
                                            L"",
                                            SOURCENAME,
                                            __LINE__,
                                            0,
                                            0,
                                            NULL,
                                            (LPCTSTR)error );
                }
        }

        return ( rc );
}



MAPPER_DB_RC
        mapper_db::do_add_static_devices (  device_list&
        device_list,

        EVENT_INDICATOR   indicator )
{
        MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

        cnfg_db& cdb = get_cnfg_db();

        CString extension;
        USHORT      type;
        ULONG plid;
        ULONG ppid;

        while ( device_list.get_next_device (extension, &type, &plid,
&ppid) )
        {
                tapi_static_line *line_p =
static_line_mgr.remove_from_free_list();
                tapi_static_phone *phone_p =
static_phone_mgr.remove_from_free_list();

                if ( (line_p != NULL) && (phone_p != NULL) )
                {
```

```
                line_p->parent_p = phone_p;
                phone_p->parent_p = line_p;

                line_p->devid = next_device_id;
                phone_p->devid = next_device_id;

            if ( indicator == EVENT_INDICATOR_SEND_EVENT )
                {
                line_p->devid |= TEMP_BIT;
                    phone_p->devid |= TEMP_BIT;
            }

                tapi_static_address *address_p = line_p-
>get_static_address();

                wcscpy (    line_p->primary_extension,
                            (LPCTSTR)extension );
                wcscpy (    phone_p->primary_extension,
                            (LPCTSTR)extension );
                wcscpy (    address_p->primary_extension,
                            (LPCTSTR)extension );

                line_p->quick_key = _wtol (
(LPCTSTR)(extension.Right(7)) );
                phone_p->quick_key = line_p->quick_key;
                address_p->quick_key = line_p->quick_key;

                lineaddresscaps_data& address_caps =
                      address_p->get_lineaddresscaps();

                address_caps.ts_lineaddresscaps.dwLineDeviceID =
                      line_p->devid + line_device_id_base;

                address_caps.ts_address = extension;

                if ( type == anate )
                {
      address_caps.ts_lineaddresscaps.dwAddrCapFlags &=
                            ~LINEADDRCAPFLAGS_ORIGOFFHOOK;
                }

                linedevcaps_data& line_caps = line_p-
>get_linedevcaps();

                line_caps.ts_linedevcaps.dwPermanentLineID = plid;

                line_caps.ts_line_name = extension;

                phonecaps_data& phone_caps = phone_p-
>get_phonecaps();

                phone_caps.ts_phonecaps.dwPermanentPhoneID = ppid;

                phone_caps.ts_phone_name = extension;

                switch ( type )
                {
```

```
                        case anate: phone_caps.ts_phone_info =
                                                _T("Analog Device");
    break;

                        case opslma:
                        case opstl: phone_caps.ts_phone_info =
                                                _T("Off Premise
    Station"); break;

                        case extvcml: phone_caps.ts_phone_info =
                                                _T("External Voice
    Mail"); break;

                        case kysetjr:
                        case kysetdy: phone_caps.ts_phone_info =
                                                _T("Keyset"); break;

                        case phantom: phone_caps.ts_phone_info =
                                                _T("Phantom"); break;

                        case rp120:
                        case rp120D: phone_caps.ts_phone_info =
                                                _T("Rolmphone 120");
    break;

                        case rp240:
                        case rp240B:
                        case rp240D:
                        case rp240E:
                        case rp240ED: phone_caps.ts_phone_info =
                                                _T("Rolmphone 120");
    break;

                        case rp312:
                        case rp312L: phone_caps.ts_phone_info =
                                                _T("Rolmphone 312");
    break;

                        case rp400:
                        case rp400D: phone_caps.ts_phone_info =
                                                _T("Rolmphone 400");
    break;

                        case rp612:
                        case rp612D:
                        case rp612DK:
                    case rp612K:
                    case rp612L:
                    case rp612LD:
                    case rp612LK:
                    case rp612LDK:
                    case rp612S:
                    case rp612SD:
                    case rp612SK:
                    case rp612SL:
                    case rp612SDK:
                    case rp612SLD:
                        case rp612SLK:
                        case rp612SLDK: phone_caps.ts_phone_info =
                                                _T("Rolmphone 612");
    break;

                        case rp624:
                        case rp624D:
                        case rp624DK:
                    case rp624K:
```

```
                    case rp624L:
                    case rp624LD:
                    case rp624LK:
                    case rp624LDK:
                    case rp624S:
                    case rp624SD:
                    case rp624SK:
                    case rp624SL:
                    case rp624SDK:
                    case rp624SLD:
                        case rp624SLK:
                        case rp624SLDK: phone_caps.ts_phone_info =
                                                _T("Rolmphone 624");
    break;

                        case rp4327:
                        case rp4327T: phone_caps.ts_phone_info =
                                                _T("Rolmphone 4327");
    break;

                        case ni1200SL: phone_caps.ts_phone_info =
                                                _T("Optiset-NI
    Phone"); break;

                        case optie3: phone_caps.ts_phone_info =
                                                _T("Optiset Entry
    Phone"); break;

                        case optie8: phone_caps.ts_phone_info =
                                                _T("Optiset Basic
    Phone"); break;

                        case optieSL: phone_caps.ts_phone_info =
                                                _T("Optiset Standard
    Phone"); break;

                        case optie1L: phone_caps.ts_phone_info =
                                                _T("Optiset Advanced
    Phone"); break;

                        case optie1SL: phone_caps.ts_phone_info =
                                                _T("Optiset Advanced
    Plus Phone"); break;

                        case rmdcm:
                        case adcm: phone_caps.ts_phone_info =
                                                _T("Data Comm
    Module"); break;

                        case set211: phone_caps.ts_phone_info =
                                                _T("Set 211 Phone");
    break;

                        case set260: phone_caps.ts_phone_info =
                                                _T("Set 260 Phone");
    break;

                        case set400: phone_caps.ts_phone_info =
                                                _T("Set 400 Phone");
    break;

                        case set500: phone_caps.ts_phone_info =
                                                _T("Set 500 Phone");
    break;

                        case set600: phone_caps.ts_phone_info =
```

```
                                                    _T("Set 600 Phone");
    break;

                        case set700: phone_caps.ts_phone_info =
                                                    _T("Set 700 Phone");
    break;

                        case other: phone_caps.ts_phone_info =
                                                    _T("Generic"); break;

                        default:

                              phone_caps.ts_phone_info
    =_T("Generic");


                              if ( cdb.debug_tracing_on (
    DBG_MAPPER_DB_BIT ) )
                              {
                                    CString error;
                                    error.Format (
    _T("mapper_db::do_add_static_devices(). Unexpected device type =
    %d"),
                                                                type );

                                    su_log_message    (     0,

          SU_DEBUG,

          SU_TRACE_NONE,
                                                                0,

          L"",

          SOURCENAME,

          __LINE__,
                                                                0,
                                                                0,

          NULL,

          (LPCTSTR)error );
                              }

                              break;
                        }


                        static_line_mgr.add_to_in_use_list ( line_p );
                        static_phone_mgr.add_to_in_use_list ( phone_p );


                        if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_OBJ_BIT )
    )
                        {
                              CString error;
                              error.Format (
    _T("mapper_db::do_add_static_devices().\n\
        Adding static line: Ptr = %lx, Object ID = %ld, Device ID = %d,
    Device base = %d, Quick key = %ld, Extension = %s, Monitor status =
    %d\n\
          With static address: Ptr = %lx, Object ID = %ld, Device ID =
    %d, Quick key = %ld, Extension = %s\n\
```

```
        With static phone: Ptr = %lx, Object ID = %ld, Device ID = %d,
Device base = %d, Quick key = %ld, Extension = %s, Monitor status =
%d"),
                                    line_p,
                                    line_p->object_id,
                                    line_p->devid,
                          line_device_id_base,
                                    line_p->quick_key,
                                    line_p->primary_extension,
                                    line_p->monitor_on,
                                    address_p,
                                    address_p->object_id,
                                    address_p->devid,
                                    address_p->quick_key,
                                    address_p->primary_extension,
                                    phone_p,
                                    phone_p->object_id,
                                    phone_p->devid,
                          phone_device_id_base,
                                    phone_p->quick_key,
                                    phone_p->primary_extension,
                                    phone_p->monitor_on );

                              su_log_message    (     0,
                                                      SU_DEBUG,
                                                      SU_TRACE_NONE,
                                                      0,
                                                      L"",
                                                      SOURCENAME,
                                                      __LINE__,
                                                      0,
                                                      0,
                                                      NULL,
                                                      (LPCTSTR)error
);
                         }

                         if ( indicator == EVENT_INDICATOR_SEND_EVENT )
                         {

                              cnfg_db& cdb = get_cnfg_db();

                              HPROVIDER ph = cdb.get_provider_handle();


                              tapi_interface& ti = get_tapi_interface();

                              SEND_TO_TAPI_RC t_rc =
                                   ti.send_line_create ( ph, line_p-
>devid, 0 );

                              if ( t_rc != SEND_TO_TAPI_RC_GOOD )
                              {
                                   CString error;
                                   error.Format (
_T("mapper_db::do_add_static_devices(). Problem sending event: line
create.") );
                                   su_log_message    (     0,
                                                           SU_DEBUG,
      SU_TRACE_NONE,
                                                           0,
                                                           L"",
      SOURCENAME,
```

```
                                                                    __LINE__,
                                                                    t_rc,
                                                                    0,
                                                                    NULL,
5
        (LPCTSTR)error );
                        }

10                      if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT
) )
                        {
                                CString error;
                                error.Format (
15  _T("mapper_db::do_add_static_devices(). Sent event: line create") );
                                su_log_message    (     0,
        SU_DEBUG,
20
        SU_TRACE_NONE,
                                                                    0,
                                                                    L"",
25      SOURCENAME,
                                                                    __LINE__,
                                                                    t_rc,
                                                                    0,
                                                                    NULL,
30
        (LPCTSTR)error );
                        }

35                      t_rc = ti.send_phone_create ( ph, phone_p-
>devid, 0 );
                        if ( t_rc != SEND_TO_TAPI_RC_GOOD )
                        {
40                              CString error;
                                error.Format (
_T("mapper_db::do_add_static_devices(). Problem sending event: phone
create.") );
                                su_log_message    (     0,
45                                                                  SU_DEBUG,
        SU_TRACE_NONE,
                                                                    0,
                                                                    L"",
50
        SOURCENAME,
                                                                    __LINE__,
                                                                    t_rc,
                                                                    0,
55                                                                  NULL,
        (LPCTSTR)error );
                        }

60
                        if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT
) )
                        {
                                CString error;
65                              error.Format (
_T("mapper_db::do_add_static_devices(). Sent event: phone create") );
```

```
                                    su_log_message    (     0,
        SU_DEBUG,
5
        SU_TRACE_NONE,
                                                                0,
                                                                L"",
10      SOURCENAME,
                                                                __LINE__,
                                                                t_rc,
                                                                0,
                                                                NULL,
15
        (LPCTSTR)error );
                                }

20                              c_stats_db& stats_db = get_stats_db();


                                stats_db.admin_stats_ptr->
                                      eng_statistics.eng_mapper_db_stats.
25                                          outstanding_line_creates++;

                                stats_db.admin_stats_ptr->
                                      eng_statistics.eng_mapper_db_stats.
                                            outstanding_phone_creates++;
30                      }


                        ++next_device_id;
                }
35              else
                {

                        rc = MAPPER_DB_RC_FAILED;

40                      if ( line_p != NULL )
                        {
                              static_line_mgr.add_to_free_list ( line_p );
                        }

45                      if ( phone_p != NULL )
                        {
                              static_phone_mgr.add_to_free_list ( phone_p
    );
                        }
50
                        CString error;
                        error.Format (
    _T("mapper_db::do_add_static_devices(). Could not allocate, line =
    %lx, phone= %lx."),
55                                          line_p, phone_p );
                        su_log_message    (     0,
                                                SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
60                                              L"",
                                                SOURCENAME,
                                                __LINE__,
                                                0,
                                                0,
65                                              NULL,
                                                (LPCTSTR)error );
```

```
                    break;
                }
        }

        return ( rc );
}



MAPPER_DB_RC
        mapper_db::get_static_device (      DWORD
        device_id,
        tapi_static_line  **line_pp ) const
{
        MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

        if ( line_pp != NULL )
        {
              DWORD sem_rc =
                    WaitForSingleObject ( mapper_db_sem,
MAPPER_DB_SEM_WAIT );

              if ( sem_rc == WAIT_OBJECT_0 )
              {
                    *line_pp =
                          static_line_mgr.find ( device_id -
line_device_id_base );

                    if ( *line_pp != NULL )
                    {
                          cnfg_db& cdb = get_cnfg_db();

                          if ( cdb.debug_tracing_on (
DBG_MAPPER_DB_OBJ_BIT ) )
                          {
                                tapi_static_address * address_p =
                                      static_cast<tapi_static_address
*>
                                            ((*line_pp)->child1_p );

                                CString error;
                                error.Format (
_T("mapper_db::get_static_device(line). Given device ID = %ld\n\
    Found static line: Ptr = %lx, Object ID = %ld, Device ID = %d,
Device base = %d, Quick key = %ld, Extension = %s, Monitor status =
%d\n\
```

```
        With static address: Ptr = %lx, Object ID = %ld, Device ID =
%d, Quick key = %ld, Extension = %s"),
                                        device_id,
                                        *line_pp,
                                        (*line_pp)->object_id,
                                        (*line_pp)->devid,
                                line_device_id_base,
                                        (*line_pp)->quick_key,
                                        (*line_pp)->primary_extension,
                                        (*line_pp)->monitor_on,
                                        address_p,
                                        address_p->object_id,
                                        address_p->devid,
                                        address_p->quick_key,
                                        address_p->primary_extension );

                                su_log_message    (     0,
                                                        SU_DEBUG,

        SU_TRACE_NONE,
                                                        0,
                                                        L"",

        SOURCENAME,
                                                        __LINE__,
                                                        0,
                                                        0,
                                                        NULL,

        (LPCTSTR)error );
                                }
                        }
                        else
                        {

                                rc = MAPPER_DB_RC_FAILED;

                                CString error;
                                error.Format (
_T("mapper_db::get_static_device(line). Could not find device, ID =
%ld "),
                                                        device_id  );
                                su_log_message    (     0,
                                                                SU_DEBUG,
                                                                SU_TRACE_NONE,
                                                                0,
                                                                L"",
                                                                SOURCENAME,
                                                                __LINE__,
                                                                0,
                                                                0,
                                                                NULL,
                                                                (LPCTSTR)error
);
                        }

                        ReleaseSemaphore ( mapper_db_sem, 1, NULL );
                }
                else
                {

                        rc = MAPPER_DB_RC_TIMEOUT;
                }
        }
        else
```

```
        {
                rc = MAPPER_DB_RC_BAD_PARAM;
                CString error;
                error.Format ( _T("mapper_db::get_static_device(line).
Bad param, device ptr = %lx "),
                                        line_pp  );
                su_log_message    (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        L"",
                                        SOURCENAME,
                                        __LINE__,
                                        0,
                                        0,
                                        NULL,
                                        (LPCTSTR)error );
        }

        return ( rc );
}


MAPPER_DB_RC
        mapper_db::get_static_device (      DWORD
        device_id,

        tapi_static_phone **phone_pp ) const
{
        MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;


        if ( phone_pp != NULL )
        {
                DWORD sem_rc =
                        WaitForSingleObject ( mapper_db_sem,
MAPPER_DB_SEM_WAIT );

                if ( sem_rc == WAIT_OBJECT_0 )
                {
                        *phone_pp =
                                static_phone_mgr.find ( device_id -
phone_device_id_base );

                        if ( *phone_pp != NULL )
                        {
                                cnfg_db& cdb = get_cnfg_db();

                                if ( cdb.debug_tracing_on (
DBG_MAPPER_DB_OBJ_BIT ) )
                                {
                                        CString error;
                                        error.Format (
_T("mapper_db::get_static_device(phone). Given device ID = %ld\n\
```

```
    Found static phone: Ptr = %lx, Object ID = %ld, Device ID = %d,
Device base = %d, Quick key = %ld, Extension = %s, Monitor status =
%d"),
                                        device_id,
                                        *phone_pp,
                                        (*phone_pp)->object_id,
                                        (*phone_pp)->devid,
                                phone_device_id_base,
                                        (*phone_pp)->quick_key,
                                        (*phone_pp)->primary_extension,
                                        (*phone_pp)->monitor_on );

                                su_log_message    (     0,
                                                        SU_DEBUG,
        SU_TRACE_NONE,
                                                        0,
                                                        L"",
        SOURCENAME,
                                                        __LINE__,
                                                        0,
                                                        0,
                                                        NULL,
        (LPCTSTR)error );
                                }
                        }
                        else
                        {
                                rc = MAPPER_DB_RC_FAILED;

                                CString error;
                                error.Format (
_T("mapper_db::get_static_device(phone). Could not find device, ID =
%ld "),
                                                        device_id  );
                                su_log_message    (     0,
                                                        SU_DEBUG,
                                                        SU_TRACE_NONE,
                                                        0,
                                                        L"",
                                                        SOURCENAME,
                                                        __LINE__,
                                                        0,
                                                        0,
                                                        NULL,
                                                        (LPCTSTR)error
);
                        }

                        ReleaseSemaphore ( mapper_db_sem, 1, NULL );
                }
                else
                {
                        rc = MAPPER_DB_RC_TIMEOUT;
                }
        }
        else
        {
                rc = MAPPER_DB_RC_BAD_PARAM;
```

```
            CString error;
            error.Format ( _T("mapper_db::get_static_device(phone).
Bad param, device ptr = %lx "),
                                    phone_pp  );
            su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          L"",
                                          SOURCENAME,
                                          __LINE__,
                                          0,
                                          0,
                                          NULL,
                                          (LPCTSTR)error );
      }

      return ( rc );
}



MAPPER_DB_RC mapper_db::create_devices ( device_list& dev_list )
{
      MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

      cnfg_db& cdb = get_cnfg_db();

      if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT ) )
      {
            CString error;
            error.Format ( _T("mapper_db:create_devices()") );

            su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          L"",
                                          SOURCENAME,
                                          __LINE__,
                                          0,
                                          0,
                                          NULL,
                                          (LPCTSTR)error );
      }

      if ( dev_list.validate() == TRUE )
      {
            DWORD sem_rc =
                  WaitForSingleObject ( mapper_db_sem,
MAPPER_DB_SEM_WAIT );
```

```
            if ( sem_rc == WAIT_OBJECT_0 )
            {

                  device_list devices;
                  devices.init();


                  tapi_static_line *line_p =
static_line_mgr.get_first();

                  while ( line_p != NULL )
                  {

                        devices.add_device ( line_p-
>primary_extension, 0, 0 );


                        line_p = line_p->get_next();
                  }


                  do_delete_static_devices ( devices,
EVENT_INDICATOR_NONE );


                  next_device_id = 0;


                  do_add_static_devices ( dev_list,
EVENT_INDICATOR_NONE );


            c_stats_db& stats_db = get_stats_db();

                stats_db.admin_stats_ptr->
                      eng_statistics.eng_mapper_db_stats.
                    avail_lines_from_pbx =
                        static_line_mgr.get_in_use_count();

            stats_db.admin_stats_ptr->
                      eng_statistics.eng_mapper_db_stats.
                    avail_phones_from_pbx =
                        static_phone_mgr.get_in_use_count();


            stats_db.admin_stats_ptr->
                tsp_stats.line_dev_max =
                    static_line_mgr.get_in_use_count();

            stats_db.admin_stats_ptr->
                tsp_stats.phone_dev_max =
                    static_phone_mgr.get_in_use_count();


                  ReleaseSemaphore ( mapper_db_sem, 1, NULL );
            }
            else
            {

                  rc = MAPPER_DB_RC_TIMEOUT;
            }
      }
      else
      {

            rc = MAPPER_DB_RC_BAD_PARAM;
```

```
            CString error;
            error.Format ( _T("mapper_db::create_devices(). Invalid
device lsit.") );
            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    L"",
                                    SOURCENAME,
                                    __LINE__,
                                    0,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
      }

      return ( rc );
}



MAPPER_DB_RC
      mapper_db::update_devices ( device_list& new_device_list )
{
      MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

      cnfg_db& cdb = get_cnfg_db();

      if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT ) )
      {
            CString error;
            error.Format ( _T("mapper_db:update_devices()") );

            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    L"",
                                    SOURCENAME,
                                    __LINE__,
                                    0,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
      }

      device_list original_device_list;
      cdb.get_device_list ( original_device_list );

      device_list added_device_list;
      device_list deleted_device_list;

      added_device_list.init();
      deleted_device_list.init();

      original_device_list.create_differences ( new_device_list,

      added_device_list,
```

```
        deleted_device_list );

        if ( (added_device_list.validate() == TRUE) ||
                (deleted_device_list.validate() == TRUE) )
        {
              DWORD sem_rc =
                    WaitForSingleObject ( mapper_db_sem,
MAPPER_DB_SEM_WAIT );

              if ( sem_rc == WAIT_OBJECT_0 )
              {
                    rc = do_add_static_devices ( added_device_list,
EVENT_INDICATOR_SEND_EVENT );
                    if ( rc == MAPPER_DB_RC_GOOD )
                    {
                          rc = do_delete_static_devices (
deleted_device_list,
        EVENT_INDICATOR_SEND_EVENT );
                      c_stats_db& stats_db = get_stats_db();

                      stats_db.admin_stats_ptr->
                            eng_statistics.eng_mapper_db_stats.
                          avail_lines_from_pbx =
                              static_line_mgr.get_in_use_count();


                      stats_db.admin_stats_ptr->
                            eng_statistics.eng_mapper_db_stats.
                          avail_phones_from_pbx =
                              static_phone_mgr.get_in_use_count();
                  stats_db.admin_stats_ptr->
                      tsp_stats.line_dev_max =
                          static_line_mgr.get_in_use_count();

                  stats_db.admin_stats_ptr->
                      tsp_stats.phone_dev_max =
                          static_phone_mgr.get_in_use_count();

                          if ( rc != MAPPER_DB_RC_GOOD )
                          {
                                CString error;
                                error.Format (
_T("mapper_db::update_devices(). Error delete devices.") );
                                su_log_message    (     0,
                                                                SU_DEBUG,
        SU_TRACE_NONE,
                                                                0,
                                                                L"",
        SOURCENAME,
                                                                __LINE__,
                                                                rc,
                                                                0,
                                                                NULL,
        (LPCTSTR)error );
```

```
                        }
                    }
                    else
                    {
                        CString error;
                        error.Format (
_T("mapper_db::update_devices(). Error deleting devices.") );
                        su_log_message    (     0,
                                                SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
                                                L"",
                                                SOURCENAME,
                                                __LINE__,
                                                0,
                                                0,
                                                NULL,
                                                (LPCTSTR)error
);
                    }

                    ReleaseSemaphore ( mapper_db_sem, 1, NULL );
                }
                else
                {

                    rc = MAPPER_DB_RC_TIMEOUT;
                }
        }

        return ( rc );
}



MAPPER_DB_RC
        mapper_db::make_line_permanent ( DWORD temp_id, DWORD device_id
)
{
        MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

        cnfg_db& cdb = get_cnfg_db();

        if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT ) )
        {
                CString error;
                error.Format ( _T("mapper_db:make_line_permanent(). Temp
id = %ld, device Id = %ld"),
                    temp_id, device_id );

                su_log_message    (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        L"",
                                        SOURCENAME,
                                        __LINE__,
```

```
                                                    0,
                                                    0,
                                                    NULL,
                                                    (LPCTSTR)error );
        }

        tapi_static_line *line_p = static_line_mgr.find ( temp_id );

        if ( line_p != NULL )
        {
          static_line_mgr.remove_from_in_use_list ( line_p );

              line_p->devid = device_id - line_device_id_base;

          static_line_mgr.add_to_in_use_list ( line_p );

              c_stats_db& stats_db = get_stats_db();

              stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                          outstanding_line_creates--;

          stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                lines_created++;

              stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                avail_lines_from_pbx =
                    static_line_mgr.get_in_use_count();

          stats_db.admin_stats_ptr->
              tsp_stats.line_dev_max =
                  static_line_mgr.get_in_use_count();

              if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_OBJ_BIT ) )
              {
                    tapi_static_address * address_p =
                          static_cast<tapi_static_address *>
                                (line_p->child1_p );

                    CString error;
                    error.Format (
_T("mapper_db::make_line_permanent().\n\
    Static line: Ptr = %lx, Object ID = %ld, Device ID = %d, Device
base = %d, Quick key = %ld, Extension = %s, Monitor status = %d\n\
     With static address: Ptr = %lx, Object ID = %ld, Device ID =
%d, Quick key = %ld, Extension = %s"),
                          line_p,
                          line_p->object_id,
                          line_p->devid,
                line_device_id_base,
                          line_p->quick_key,
                          line_p->primary_extension,
                          line_p->monitor_on,
                          address_p,
```

```
                        address_p->object_id,
                        address_p->devid,
                        address_p->quick_key,
                        address_p->primary_extension );

                su_log_message    (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        L"",
                                        SOURCENAME,
                                        __LINE__,
                                        0,
                                        0,
                                        NULL,
                                        (LPCTSTR)error );
            }
        }
        else
        {
            rc = MAPPER_DB_RC_FAILED;

            CString error;
            error.Format ( _T("mapper_db::make_line_permanent().
Could find temporary line, temp ID = %ld, TAPI id = %ld."),
                                    temp_id, device_id );
            su_log_message    (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        L"",
                                        SOURCENAME,
                                        __LINE__,
                                        0,
                                        0,
                                        NULL,
                                        (LPCTSTR)error );
        }

        return ( rc );
}


MAPPER_DB_RC
        mapper_db::make_phone_permanent ( DWORD temp_id, DWORD
device_id )
{
        MAPPER_DB_RC rc = MAPPER_DB_RC_GOOD;

        cnfg_db& cdb = get_cnfg_db();

        if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_BIT ) )
        {
            CString error;
            error.Format ( _T("mapper_db:make_phone_permanent(). Temp
id = %ld, device Id = %ld"),
                    temp_id, device_id );

            su_log_message    (     0,
                                        SU_DEBUG,
```

```
                                                SU_TRACE_NONE,
                                                0,
                                                L"",
                                                SOURCENAME,
                                                __LINE__,
                                                0,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
        }

        tapi_static_phone *phone_p = static_phone_mgr.find ( temp_id );

        if ( phone_p != NULL )
        {
          static_phone_mgr.remove_from_in_use_list ( phone_p );

              phone_p->devid = device_id - phone_device_id_base;

          static_phone_mgr.add_to_in_use_list ( phone_p );

              c_stats_db& stats_db = get_stats_db();

              stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                          outstanding_phone_creates--;

          stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                phones_created++;

              stats_db.admin_stats_ptr->
                    eng_statistics.eng_mapper_db_stats.
                avail_phones_from_pbx =
                    static_phone_mgr.get_in_use_count();

          stats_db.admin_stats_ptr->
              tsp_stats.phone_dev_max =
                  static_phone_mgr.get_in_use_count();

              if ( cdb.debug_tracing_on ( DBG_MAPPER_DB_OBJ_BIT ) )
              {
                    CString error;
                    error.Format (
_T("mapper_db::make_phone_permanent().\n\
    Deleting static phone: Ptr = %lx, Object ID = %ld, Device ID =
%d, Device base = %d, Quick key = %ld, Extension = %s, Monitor status
= %d"),
                          phone_p,
                          phone_p->object_id,
                          phone_p->devid,
                phone_device_id_base,
                          phone_p->quick_key,
                          phone_p->primary_extension,
                          phone_p->monitor_on );
```

```
                su_log_message    (     0,
                                                        SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
                                                L"",
                                                SOURCENAME,
                                                __LINE__,
                                                0,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
                }
        }
        else
        {
                rc = MAPPER_DB_RC_FAILED;

                CString error;
                error.Format ( _T("mapper_db::make_phone_permanent().
Could find temporary phone, temp ID = %ld, TAPI id = %ld."),
                                                temp_id, device_id );
                su_log_message    (     0,
                                                SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
                                                L"",
                                                SOURCENAME,
                                                __LINE__,
                                                0,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
        }

        return ( rc );
}


enum MAPPER_DB_RC
        {       MAPPER_DB_RC_GOOD                       = 0,
                MAPPER_DB_RC_FAILED                             = 1,
                MAPPER_DB_RC_BAD_PARAM                  = 2,
                MAPPER_DB_RC_NOT_FOUND                  = 3,
                MAPPER_DB_RC_TIMEOUT                    = 4,
                MAPPER_DB_RC_DEVICE_NOT_OPEN    = 5,
                MAPPER_DB_RC_OBJECT_INVALID             = 6 };


enum EVENT_INDICATOR
        {       EVENT_INDICATOR_NONE            = 0,
                EVENT_INDICATOR_SEND_EVENT      = 1 };

const ULONG TEMP_BIT = 0x80000000;

#define MAPPER_DB_SEM   _T("MAPPER_DB_SEM")

const MAPPER_DB_SEM_WAIT = 2000; // 2 second max wait
```

```
enum MONITOR_STATUS
      {     MONITORS_LOST          = 0,
            MONITORS_NOT_LOST = 1 };


class mapper_db
{
public:

      friend mapper_db& get_mapper_db();

      friend class static_line_manager;
      friend class open_line_manager;

private:
            mapper_db();

      BOOL valid;


      HANDLE mapper_db_sem;

      static_phone_manager    static_phone_mgr;
      static_line_manager           static_line_mgr;
      open_phone_manager            open_phone_mgr;
      open_line_manager       open_line_mgr;
      request_id_manager            request_id_mgr;
      call_manager                  call_mgr;
      static_address_manager  static_address_mgr;
      address_manager               address_mgr;


      DWORD line_device_id_base;
      DWORD phone_device_id_base;
      DWORD next_device_id;


      MAPPER_DB_RC do_async_failures ( tapi_object * );
      MAPPER_DB_RC do_delete_call ( tapi_call * );
      MAPPER_DB_RC do_delete_open_phone_device ( tapi_open_phone * );
      MAPPER_DB_RC do_delete_open_line_device ( tapi_open_line * );
      MAPPER_DB_RC do_delete_static_devices( device_list&,
EVENT_INDICATOR);
      MAPPER_DB_RC do_add_static_devices ( device_list&,
EVENT_INDICATOR );

public:


      ~mapper_db();


      BOOL                    validate() const;
      BOOL                    validate ( const tapi_open_line *,
ULONG=0 ) const;
      BOOL                    validate ( const tapi_open_phone *,
ULONG=0 ) const;
      BOOL                    validate ( const tapi_call *, ULONG=0 )
const;
      tapi_open_line *  handle_to_object ( HDRVLINE ) const;
      tapi_open_phone *       handle_to_object ( HDRVPHONE ) const;
      tapi_call *             handle_to_object ( HDRVCALL ) const;
      HDRVLINE                object_to_handle ( const tapi_open_line
* ) const;
```

```
      HDRVPHONE               object_to_handle ( const
tapi_open_phone *) const;
      HDRVCALL                object_to_handle ( const tapi_call * )
const;

      void        set_device_id_base ( DWORD, DWORD );
      void        get_device_id_base ( DWORD *, DWORD * ) const;
      USHORT            get_num_line_devices() const;
      USHORT            get_num_phone_devices() const;

      MAPPER_DB_RC create_open_device ( DWORD, tapi_open_line **,
ULONG * );
      MAPPER_DB_RC create_open_device ( DWORD, tapi_open_phone **,
ULONG * );
      MAPPER_DB_RC delete_open_device ( tapi_open_device ** );
      MAPPER_DB_RC get_static_device ( DWORD, tapi_static_line ** )
const;
      MAPPER_DB_RC get_static_device ( DWORD, tapi_static_phone ** )
const;
      MAPPER_DB_RC get_open_device ( const CString&, tapi_open_line
**, tapi_address ** ) const;
      MAPPER_DB_RC get_open_device ( const CString&, tapi_open_phone
** ) const;
      MAPPER_DB_RC lock_open_device ( tapi_open_device *, ULONG=2000
);
      MAPPER_DB_RC unlock_open_device ( tapi_open_device * );
      MAPPER_DB_RC create_call ( tapi_address *, tapi_call **,
                                                ULONG *,

      EVENT_INDICATOR=EVENT_INDICATOR_NONE );
      MAPPER_DB_RC delete_call ( tapi_call ** );
      MAPPER_DB_RC move_call_to_old_call_list ( tapi_call * );
      MAPPER_DB_RC create_request_id ( USHORT, DWORD,

      tapi_open_device *,
                                                              tapi_call
* = NULL );
      MAPPER_DB_RC delete_request_id ( USHORT, DWORD *,

      tapi_open_device **,
                                                              tapi_call
** );
      MAPPER_DB_RC create_devices ( device_list& );
      MAPPER_DB_RC update_devices ( device_list& );

      MAPPER_DB_RC close_all_devices();
      MAPPER_DB_RC process_link_down();
      MAPPER_DB_RC process_link_up ( MONITOR_STATUS );
      MAPPER_DB_RC process_messages_lost();

      MAPPER_DB_RC make_line_permanent ( DWORD, DWORD );
      MAPPER_DB_RC make_phone_permanent ( DWORD, DWORD );
};


#include "stdafx.h"

#include "tapi.h"
#include "tspi.h"
```

```
#include "mu_devls.h"

#include "su.h"
DEFINESOURCEINFO;
#include "su_shdat.h"
#include "cdb_main.h"
#include "sh_pl_ad.h"


device_list::device_list()
: valid(FALSE), device_list_p(NULL)
{
      init();
}


device_list::~device_list()
{
      delete [] device_list_p;
}



void device_list::init ( ULONG num_entries )
{
      valid = FALSE;

      delete [] device_list_p;

      device_list_p = new DEVICE_LIST_ENTRY[num_entries];
      if ( device_list_p != NULL )
      {
            num_device_list_entries = 0;
            max_device_list_entry = num_entries - 1;
            next_device_list_entry = 0;

            valid = TRUE;
      }
}


BOOL device_list::validate() const
{
      if ( valid == FALSE )
      {
            CString error;
            error.Format ( _T("Device list object is invalid") );
```

```
            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    _T(""),
                                    SOURCENAME,
                                    __LINE__,
                                    (USHORT)valid,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
        }

        return ( valid );
}


ULONG device_list::get_num_entries() const
{
    ULONG num_entries = 0;

      if ( valid  )
      {
         num_entries = num_device_list_entries;
      }
      else
        {
            CString error;
            error.Format ( _T("Device list object is invalid") );
            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    _T(""),
                                    SOURCENAME,
                                    __LINE__,
                                    (USHORT)valid,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
        }

        return ( num_entries );
}


BOOL device_list::entry_in_object ( DEVICE_LIST_ENTRY *entry_p )
const
{
        BOOL rc = FALSE;

        if ( valid )
        {
            for ( ULONG i = 0; i < num_device_list_entries; ++i )
```

```
            {
                  DEVICE_LIST_ENTRY *dp;
                  dp = device_list_p + i;
                  if ( wcscmp( dp->extension, entry_p->extension) ==
0 )
                  {
                        rc = TRUE;
                        break;
                  }
            }
      }
      else
      {
            CString error;
            error.Format ( _T("Device list object is invalid") );
            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    _T(""),
                                    SOURCENAME,
                                    __LINE__,
                                    (USHORT)rc,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
      }

      return ( rc );
}


BOOL device_list::add_device (      const CString& extension,
                                          USHORT device_type,
                                          USHORT skip_digits )
{
      BOOL rc = FALSE;

      if ( (valid == TRUE) && (num_device_list_entries >
max_device_list_entry) )
      {
            DEVICE_LIST_ENTRY *temp_p =
                  new DEVICE_LIST_ENTRY[(max_device_list_entry + 1) +
ENTRIES_TO_STEP];

            if ( temp_p != 0 )
            {
                  memcpy (    temp_p,
                              device_list_p,
                              sizeof(DEVICE_LIST_ENTRY) *
(max_device_list_entry + 1) );

                  max_device_list_entry += ENTRIES_TO_STEP;
```

```
                    delete [] device_list_p;
                    device_list_p = temp_p;
                    rc = TRUE;
              }
              else
              {
                    rc = FALSE;
                    CString error;
                    error.Format ( _T("Could not allocate system
memory") );
                    su_log_message    (     0,
                                            SU_DEBUG,
                                            SU_TRACE_NONE,
                                            0,
                                            _T(""),
                                            SOURCENAME,
                                            __LINE__,
                                            (USHORT)rc,
                                            0,
                                            NULL,
                                            (LPCTSTR)error );
              }
        }
        else if ( valid == TRUE )
        {
              rc = TRUE;
        }
        else
        {
              CString error;
              error.Format ( _T("Device list object is invalid") );
              su_log_message    (     0,
                                            SU_DEBUG,
                                            SU_TRACE_NONE,
                                            0,
                                            _T(""),
                                            SOURCENAME,
                                            __LINE__,
                                            (USHORT)rc,
                                            0,
                                            NULL,
                                            (LPCTSTR)error );
        }

        if ( rc == TRUE )
        {
              memcpy (    &(device_list_p + num_device_list_entries)-
>extension,
                                  (LPCTSTR)extension,
                                  (extension.GetLength() + 1) *
sizeof(TCHAR) );
              (device_list_p + num_device_list_entries)->skip_digits =
skip_digits;
```

```
            (device_list_p + num_device_list_entries)->device_type =
device_type;

            ++num_device_list_entries;
      }

      cnfg_db& cdb = get_cnfg_db();

      if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
      {
            CString error;
            error.Format ( _T("Device List::add_device(). Extension =
%s, skip digits = %d, device type = %d"),
                                          extension, skip_digits,
device_type );

            su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          _T(""),
                                          SOURCENAME,
                                          __LINE__,
                                          (USHORT)rc,
                                          0,
                                          NULL,
                                          (LPCTSTR)error );
      }

      return ( rc );
}

BOOL device_list::get_next_device ( CString&    extension,
                                                USHORT
      *device_type_p,
                                                ULONG
      *perm_line_id_p,
                                                ULONG
      *perm_phone_id_p )
{
      BOOL rc = FALSE;

      USHORT      device_type = 0;
      ULONG perm_line_id = 0;
      ULONG perm_phone_id = 0;

      if ( (valid == TRUE) && (device_type_p != NULL) &&
            (perm_line_id_p != NULL) && (perm_phone_id_p != NULL) )
      {
            if ( next_device_list_entry < num_device_list_entries )
            {
                  extension = (device_list_p +
next_device_list_entry)->extension;
                  device_type = (device_list_p +
next_device_list_entry)->device_type;

                                    USHORT skip = (device_list_p +
next_device_list_entry)->skip_digits;
```

```
#ifdef UNICODE
                    perm_line_id = _wtol ( (LPCTSTR)extension + skip );
#else
                    perm_line_id = atol ( (LPCTSTR)extension + skip );
#endif
                    perm_phone_id = perm_line_id;


                    *device_type_p = device_type;
                    *perm_line_id_p = perm_line_id;
                    *perm_phone_id_p = perm_phone_id;


                    ++next_device_list_entry;

                    rc = TRUE;
               }
          }
          else
          {
               CString error;
               error.Format ( _T("Device list not accessible. Device
type ptr = %lx, line ID ptr = %lx, phone ID ptr = %lx"),
                                        device_type_p, perm_line_id_p,
perm_phone_id_p );
               su_log_message     (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        _T(""),
                                        SOURCENAME,
                                        __LINE__,
                                        (USHORT)valid,
                                        0,
                                        NULL,
                                        (LPCTSTR)error );
          }

          cnfg_db& cdb = get_cnfg_db();

          if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
          {
               CString error;
               error.Format ( _T("Device List::get_next_device().
Extension = %s, device type = %ld, perm line id = %ld, perm phone id
= %ld"),
                                        extension, device_type,
perm_line_id, perm_phone_id );
               su_log_message     (     0,
                                        SU_DEBUG,
                                        SU_TRACE_NONE,
                                        0,
                                        _T(""),
                                        SOURCENAME,
                                        __LINE__,
                                        (USHORT)rc,
                                        0,
                                        NULL,
                                        (LPCTSTR)error );
          }
```

```
        return ( rc );
}


BOOL device_list::create_differences (    const device_list&
new_list,
        device_list& added_list,
        device_list& deleted_list ) const
{
        BOOL rc = FALSE;

        added_list.init();
        deleted_list.init();

        if (   (valid == TRUE) &&
                    (added_list.validate() == TRUE) &&
                    (deleted_list.validate() == TRUE) &&
                    (new_list.validate() == TRUE) )
        {
                rc = TRUE;

                for ( ULONG i = 0; i < new_list.num_device_list_entries;
++i)
                {
                    if ( !(entry_in_object( new_list.device_list_p +
i)) )
                    {
                            DEVICE_LIST_ENTRY *dp =
new_list.device_list_p + i;

                            rc = added_list.add_device (  dp->extension,
        dp->device_type,
        dp->skip_digits );

                            if ( rc == FALSE )
                            {
                                CString error;
                                error.Format ( _T("Error adding device
to list") );

                                su_log_message    (     0,
                                                        SU_DEBUG,
                                                        SU_TRACE_NONE,
                                                        0,
                                                        _T(""),
                                                        SOURCENAME,
                                                        __LINE__,
                                                        (USHORT)rc,
                                                        0,
                                                        NULL,
                                                        (LPCTSTR)error
);
```

```
                                    break;
                              }
                        }
                  }

                  if ( rc != FALSE )
                  {
                        for ( i = 0; i < num_device_list_entries; ++i )
                        {
                              if ( !(new_list.entry_in_object(
device_list_p + i)) )
                              {
                                    DEVICE_LIST_ENTRY *dp = device_list_p +
i;

                                    rc = deleted_list.add_device (      dp-
>extension,
                  dp->device_type,
                  dp->skip_digits );

                                    if ( rc == FALSE )
                                    {
                                          CString error;
                                          error.Format ( _T("Error adding
device to list") );
                                          su_log_message    (     0,
                                                                  SU_DEBUG,
        SU_TRACE_NONE,
                                                                  0,
                                                                  _T(""),
        SOURCENAME,
                                                                  __LINE__,
        (USHORT)rc,
                                                                  0,
                                                                  NULL,
        (LPCTSTR)error );
                                          break;
                                    }
                              }
                        }
                  }
            }
            else
            {
                  CString error;
                  error.Format ( _T("A device list object is invalid,
current = %d, added = %d, deleted = %d, new = %d"),
                                    valid, added_list.validate(),
                                    deleted_list.validate(),
                                    new_list.validate() );
                  su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          _T(""),
```

```
                                                SOURCENAME,
                                                __LINE__,
                                                (USHORT)rc,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
        }

        cnfg_db& cdb = get_cnfg_db();

        if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
        {
              CString error;
              error.Format ( _T("Device List::create_differences().")
);

              su_log_message    (     0,
                                                SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
                                                _T(""),
                                                SOURCENAME,
                                                __LINE__,
                                                (USHORT)rc,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
        }

        return ( rc );
}


ULONG device_list::get_raw_format_size() const
{
        ULONG size = 0;
        if ( valid == TRUE )
        {
              size = sizeof(num_device_list_entries) +
                                (num_device_list_entries *
sizeof(DEVICE_LIST_ENTRY));
        }
        else
        {
              CString error;
              error.Format ( _T("Device list object is invalid") );

              su_log_message    (     0,
                                                SU_DEBUG,
                                                SU_TRACE_NONE,
                                                0,
                                                _T(""),
                                                SOURCENAME,
                                                __LINE__,
                                                (USHORT)valid,
                                                0,
                                                NULL,
                                                (LPCTSTR)error );
        }
```

```
        cnfg_db& cdb = get_cnfg_db();

        if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
        {
              CString error;
              error.Format ( _T("Device List::get_raw_format_size().
Size = %ld"),
                                    size );

              su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          _T(""),
                                          SOURCENAME,
                                          __LINE__,
                                          0,
                                          0,
                                          NULL,
                                          (LPCTSTR)error );
        }

        return ( size );
}


ULONG device_list::get_device_list_in_raw_format ( ULONG size,
           BYTE *data_p ) const
{
        ULONG bytes_copied = 0;

        ULONG data_size = get_raw_format_size();

        if ( ( valid == TRUE) && (data_p != NULL) && (size >=
data_size) )
        {
              *(ULONG *)data_p = num_device_list_entries;

              memcpy (    (data_p + sizeof(num_device_list_entries)),
                                device_list_p,
                                data_size -
sizeof(num_device_list_entries) );

              bytes_copied = data_size;
        }
        else if ( valid == FALSE )
        {
              CString error;
              error.Format ( _T("Device list object is unusable.
Requested data ptr = %lx, size = %ld"),
                                    data_p, size );

              su_log_message    (     0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
```

```
                                                    _T(""),
                                                    SOURCENAME,
                                                    __LINE__,
                                                    (USHORT)valid,
                                                    0,
                                                    NULL,
                                                    (LPCTSTR)error );
        }

        cnfg_db& cdb = get_cnfg_db();

        if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
        {
              CString error;
              error.Format ( _T("Device
List::get_device_list_in_raw_format(). Request pointer = %lx, request
size = %ld, bytes copied = %ld"),
                                          data_p, size, bytes_copied );

              su_log_message     (      0,
                                          SU_DEBUG,
                                          SU_TRACE_NONE,
                                          0,
                                          _T(""),
                                          SOURCENAME,
                                          __LINE__,
                                          0,
                                          0,
                                          NULL,
                                          (LPCTSTR)error );
        }

        return ( bytes_copied );
}


BOOL device_list::set_device_list_from_raw_format ( const BYTE
*data_p )
{
        ULONG data_size = 0;

        valid = FALSE;

        if ( data_p != NULL )
        {
              this->init ( *(ULONG *)data_p );

              if ( valid )
              {
                    num_device_list_entries = *(ULONG *)data_p;

                    data_size = num_device_list_entries *
sizeof(DEVICE_LIST_ENTRY);
                    memcpy (    device_list_p,
```

```
                                    (data_p +
sizeof(num_device_list_entries)),
                                    data_size );
            }
      }
      else
      {
            CString error;
            error.Format ( _T("Device list object is invalid") );

            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    _T(""),
                                    SOURCENAME,
                                    __LINE__,
                                    (USHORT)valid,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
      }


      cnfg_db& cdb = get_cnfg_db();

      if ( cdb.debug_tracing_on ( DBG_DEVICE_LIST_BIT ) )
      {
            CString error;
            error.Format ( _T("Device
List::set_device_list_from_raw_format(). Request pointer = %lx,  size
of list= %ld"),
                                    data_p, data_size );

            su_log_message    (     0,
                                    SU_DEBUG,
                                    SU_TRACE_NONE,
                                    0,
                                    _T(""),
                                    SOURCENAME,
                                    __LINE__,
                                    0,
                                    0,
                                    NULL,
                                    (LPCTSTR)error );
      }

      return ( valid );
}

const USHORT EXTENSION_LENGTH = 17;

struct DEVICE_LIST_ENTRY
{
      WCHAR             extension[EXTENSION_LENGTH];
      USHORT                  skip_digits;
      USHORT                  device_type;

};

const USHORT INITIAL_ENTRIES = 1000;      // Entries to start with
```

```
const USHORT ENTRIES_TO_STEP = 1000;       // Alloc this many more
when out


class device_list
{
      private:
                        BOOL valid;


            DEVICE_LIST_ENTRY       *device_list_p;
            ULONG num_device_list_entries;                    ULONG
      max_device_list_entry;
            ULONG next_device_list_entry;

            BOOL  entry_in_object ( DEVICE_LIST_ENTRY * ) const;

      public:


            device_list();
            ~device_list();

                        void init ( ULONG num_entries =
INITIAL_ENTRIES );

            BOOL validate() const;

        ULONG get_num_entries() const;

            BOOL add_device ( const CString&, USHORT, USHORT );
            BOOL get_next_device ( CString&, USHORT *, ULONG *, ULONG
* );

            BOOL create_differences (     const device_list&,
                                          device_list&,
                                          device_list& )
const;

            ULONG get_raw_format_size() const;
            ULONG get_device_list_in_raw_format ( ULONG, BYTE * )
const;
            BOOL set_device_list_from_raw_format ( const BYTE * );
};


//
#include "stdafx.h"
#include <afxmt.h>
#include "tapi.h"
#include "tspi.h"

#include "mu_devls.h"
#include "mdb_opti.h"
#include "mdb_tdat.h"
#include "mdb_siem.h"
#include "mdb_tobj.h"
#include "mdb_mgrs.h"
#include "mdb_mdb.h"
#include "ti_dbapi.h"

#include "ti_build.h"
```

```
#include "su.h"

#include "su_shdat.h"
#include "cdb_main.h"
#include "sh_pl_ad.h"
#include "su_tcorr.h"
#include "ti_intf.h"
#include "oss_acl.h"

#include "sh_stats.h"

#include "mu_aclm.h"

#include "mu_buffm.h"

#define INCL_BASE
#include "su.h"
#include "mpcommon.h"
#include "ugglobal.h"
#include "ugstats.h"
#include "ugerror.h"
#include "asn1code.h"
#include "oss_csta.h"

#include "mp_defs.h"
#include "rr_defs.h"
#include "evdefs.h"
#include "pbaclutl.h"
#include "pbacl_rr.h"
#include "roseprot.h"

#include "plglobal.h"

#include "rrglobal.h"
#include "rr_platf.h"
#include "rr_inc.h"
#include "rrrose.h"

char acl_signon_password[] = "CSTAGW";

DEFINESOURCEINFO;


RET_CODE mp_get_acl_devices_list(
    MP_DEVICES_LIST_REQUEST_TYPE request_type)
{
    static const CString                func_name =
_T("mp_get_acl_devices_list()");
    RET_CODE                            rc = GOOD;
    CNFG_DB_RC                          cf_rc = CNFG_DB_RC_GOOD;
    MAPPER_DB_RC                        db_rc = MAPPER_DB_RC_GOOD;
    BUFF_MGR_RC                         buff_rc = BUFF_MGR_RC_GOOD;
    WCHAR
trace_buff[MAX_BYTES_PER_LOG_ENTRY];
    SAVEDATA                            saved_data = {0};
    DWORD                               refid_timeout =
ACL_DEVICES_LIST_REQ_TIMEOUT;
    MP_BUFFER                           *acl_msg_p = NULL;
    ReferenceNumber                     *acl_ref_num_p = NULL;
    Get_Switch_Fn_Dev_Request           *acl_get_devices_list_rq_p =
NULL;
    device_list                         dev_list;
```

```
BOOL                                send_acl_request = FALSE;

cnfg_db& cf_db = get_cnfg_db();

if (cf_db.debug_tracing_on(DBG_RQST_MAPPER_BIT) == TRUE)
{
    wsprintf(trace_buff, _T("Entering %s"), func_name);
    su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
        SOURCENAME, __LINE__, 0, 0, NULL, trace_buff);
}

mapper_db& mp_db = get_mapper_db();
buffer_mgr &buffer_manager = get_acl_buffer_mgr();

    cf_rc = cf_db.get_device_list(dev_list);

    if (cf_rc == CNFG_DB_RC_GOOD &&
        dev_list.validate() == TRUE &&
        dev_list.get_num_entries() != 0)
    {
        db_rc = mp_db.create_devices(dev_list);
        if (db_rc == MAPPER_DB_RC_GOOD)
        {
            if (dev_list.get_num_entries() == 0)
            {
                send_acl_request = TRUE;
            }
            else
            {
                rc = GOOD;
                send_acl_request = FALSE;
            }
        }
        else
        {
            su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
                SOURCENAME, __LINE__, db_rc, 0, NULL,
                _T("creat_devices returned error"));
            rc = GOOD;
            send_acl_request = TRUE;
        }
    }
    else
    {
        send_acl_request = TRUE;
    }
}
else if (request_type == CHECK_DEVLIST_REQUEST)
{

    cf_rc = cf_db.get_device_list(dev_list);

    if (cf_rc == CNFG_DB_RC_GOOD &&
        dev_list.validate() == TRUE &&
        dev_list.get_num_entries() != 0)
    {
        db_rc = mp_db.create_devices(dev_list);
        if (db_rc == MAPPER_DB_RC_GOOD)
```

```
                {
                    rc = GOOD;
                }
                else
                {

                    su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
                        SOURCENAME, __LINE__, db_rc, 0, NULL,
                        _T("creat_devices returned error"));
                    rc = ERR_NUM_RR_ERROR;
                }
            }
            else
            {

                rc = ERR_NUM_RR_ERROR;
            }
        }
        else if (request_type == ADMIN_REQUEST)
        {
            send_acl_request = TRUE;
        }
        else
        {
            su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
                SOURCENAME, __LINE__, request_type, 0, NULL,
                _T("Parameter Error - request_type"));
            rc = ERR_NUM_RR_ERROR;
        }

        if (rc == GOOD && send_acl_request == TRUE)
        {
            buff_rc = buffer_manager.get_buffer((BYTE **) &acl_msg_p);
            if (buff_rc != BUFF_MGR_RC_GOOD)
            {
                su_log_message(0, SU_ERROR, SU_TRACE_NONE, 0, NULL,
                    SOURCENAME, __LINE__, buff_rc, 0, NULL,
                    _T("get_buffer returned error"));
                rc = ERR_NUM_RR_ERROR;
            }

                if (rc == GOOD)
            {
                rc = (USHORT) get_acl_request_struct_pointer(
                    gt_Switch_Fn_Dev_Request_chosen,
                    (ULONG **) &acl_get_devices_list_rq_p,
                    &acl_ref_num_p,
                    &acl_msg_p->mp_struct.u.acl_struct.acl_services);
                if (rc != GOOD)
                {
                    su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
                        SOURCENAME, __LINE__, rc, 0, NULL,
                        _T("get_acl_request_struct_pointer returned
error"));
                    rc = ERR_NUM_RR_ERROR;
                }
            }

            if (rc == GOOD)
            {

                memset(acl_get_devices_list_rq_p, 0,
sizeof(Get_Switch_Fn_Dev_Request));
```

```
            acl_get_devices_list_rq_p->getSwitchingFunctionDevices =
                getSwitchingFuncDevices;
            acl_get_devices_list_rq_p->aclDeviceType = aclstation;

            acl_msg_p->mp_struct.choice = ACL_STRUCT_CHOSEN;
            acl_msg_p->mp_struct.u.acl_struct.acl_services.choice =
                invokeService_chosen;
            acl_msg_p->mp_header.dest_link_id = ACL_LINK;
            acl_msg_p->mp_header.dest_port_id = NO_TAPI_PORT;
            acl_msg_p->mp_header.source_link_id = CSTA_LINK;
            acl_msg_p->mp_header.source_port_id = 0;
            acl_msg_p->mp_header.message_type =
A_GET_SWTCH_FUNC_DEV_REQ_STRUCT;
            acl_msg_p->mp_header.message_length = sizeof (MP_STRUCT);

            saved_data.tspi_function_type = 0;          // not used
            saved_data.open_device_object_p = NULL;     // not used
            saved_data.unique_open_device_id = 0;       // not used
            saved_data.call_object_p = NULL;            // not used
            saved_data.unique_call_object_id = 0;       // not used
            saved_data.current_callid = 0;              // not used
            saved_data.trace_correlator = 0;            // not used
            saved_data.buffer_p = (BYTE *) acl_msg_p;
            saved_data.device_request_type = request_type;
        }

        if (rc == GOOD)
        {
            rc = rose_send_acl_req(
                &acl_msg_p->mp_header,
                &acl_msg_p->mp_struct,
                &saved_data,
                refid_timeout,
                NULL,
                NULL,
                NULL);
            if (rc != GOOD)
            {
                su_log_message(0, SU_ERROR, SU_TRACE_NONE, 0, NULL,
                    SOURCENAME, __LINE__, rc, 0, NULL,
                    _T("rose_send_acl_req returned error"));
                rc = ERR_NUM_RR_ERROR;
            }
            else
            {
                rc = ERR_NUM_RR_SENT_OK;            }
        }
    }

    if (rc != ERR_NUM_RR_SENT_OK && acl_msg_p != NULL)
    {
        buff_rc = buffer_manager.free_buffer((BYTE *) acl_msg_p);
        if (buff_rc != GOOD)
        {
            su_log_message(0, SU_ERROR, SU_TRACE_NONE, 0, NULL,
                SOURCENAME, __LINE__, buff_rc, 0, NULL,
                _T("free_buffer returned error"));
            rc = ERR_NUM_RR_ERROR;
        }
    }

    if (cf_db.debug_tracing_on(DBG_RQST_MAPPER_BIT) == TRUE)
    {
        wsprintf(trace_buff, _T("Exiting %s"), func_name);
```

```
            su_log_message(0, SU_DEBUG, SU_TRACE_NONE, rc, NULL,
                SOURCENAME, __LINE__, 0, 0, NULL, trace_buff);
        }

        return(rc);
    }

    #include "stdafx.h"
    #include <afxmt.h>
    #include "tapi.h"
    #include "tspi.h"

    #include "mu_devls.h"


    #include "mdb_opti.h"
    #include "mdb_tdat.h"
    #include "mdb_siem.h"
    #include "mdb_tobj.h"
    #include "mdb_mgrs.h"
    #include "mdb_mdb.h"

    #include "ti_dbapi.h"


    #include "ti_build.h"

    #include "su.h"

    #include "su_shdat.h"
    #include "cdb_main.h"
    #include "sh_pl_ad.h"

    #include "su_tcorr.h"

    #include "ti_intf.h"

    #include "oss_acl.h"
    #include "sh_stats.h"

    #include "mu_aclm.h"

    #include "mu_buffm.h"
    #include "su.h"
    #include "mpcommon.h"
    #include "ugglobal.h"
    #include "ugstats.h"
    #include "ugerror.h"

    #include "asn1code.h"
    #include "oss_csta.h"

    #include "mp_defs.h"
    #include "rr_defs.h"
    #include "evdefs.h"
    #include "pbaclutl.h"
    #include "pbacl_rr.h"
    #include "roseprot.h"

    #include "rr_inc.h"

    #include "plglobal.h"

    DEFINESOURCEINFO;

    RET_CODE mp_map_acl_devices_list_resp(
```

```
    MP_HEADER *in_header_p,
    SAVEDATA *saved_data_p,
    MP_STRUCT *in_msg_p)
{
    static const CString                    function_name =
_T("mp_map_acl_devices_list_resp");
    RET_CODE                                rc = GOOD;
    BOOL                                    bool_rc = TRUE;
    BUFF_MGR_RC                             buff_rc = BUFF_MGR_RC_GOOD;
    CNFG_DB_RC                              cf_rc = CNFG_DB_RC_GOOD;
    MAPPER_DB_RC                            db_rc = MAPPER_DB_RC_GOOD;
    USER_REQID_MGR_RC                       ref_rc =
USER_REQID_MGR_RC_GOOD;
    WCHAR
trace_buff[MAX_BYTES_PER_LOG_ENTRY];
    ReferenceNumber                         *acl_ref_num_p = NULL;
    AclServices                             *acl_msg_p = NULL;
    Get_Switch_Fn_Dev_Response              *acl_devices_list_rs_p =
NULL;
    USHORT                                  index=0;
    DeviceConfigRecords_                    *next_dev_p = NULL;

    cnfg_db& cf_db = get_cnfg_db();

    if (cf_db.debug_tracing_on(DBG_RQST_MAPPER_BIT) == TRUE)
    {
        wsprintf(trace_buff, _T("Entering %s\n"), function_name);
        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
            SOURCENAME, __LINE__, 0, 0, NULL, trace_buff);
    }

    mapper_db& mp_db = get_mapper_db();


    if (in_header_p == NULL)
    {
        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
            SOURCENAME, __LINE__, 0, 0, NULL,
            _T("Bad input parameter - in_header_p"));
        rc = RR_MAPPING_FAILURE;
    }

    if (saved_data_p == NULL)
    {
        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
            SOURCENAME, __LINE__, 0, 0, NULL,
            _T("Bad input parameter - saved_data_p"));
        rc = RR_MAPPING_FAILURE;
    }

    if (in_msg_p == NULL)
    {
        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
            SOURCENAME, __LINE__, 0, 0, NULL,
            _T("Bad input parameter - in_msg_p"));
        rc = RR_MAPPING_FAILURE;
    }

    if (rc == GOOD)
    {

        acl_msg_p = &(in_msg_p->u.acl_struct.acl_services);

                rc = get_acl_response_struct_pointer(
            gt_Swtch_Fn_Dev_Response_chosen,
            (ULONG **) &acl_devices_list_rs_p,
```

```
            &acl_ref_num_p,
            acl_msg_p);

        if (rc == GOOD)
        {
            if (acl_devices_list_rs_p->acknowledgeCode ==
positiveAck)
            {

                static device_list dev_list;

                if (acl_devices_list_rs_p->sttnCnfgBlckNmbr == 0)
                {

                    dev_list.init();
                }

                next_dev_p = acl_devices_list_rs_p-
>deviceConfigRecords;
                index = 0;
                while (next_dev_p != NULL)
                    {
                    bool_rc = dev_list.add_device(
                        next_dev_p-
>value.deviceConfigSet.localParty.u.
                        LclPrty_extensionNumber,
                        next_dev_p->value.DvcCnfgRcrd_aclDvcTyp,
                        next_dev_p->value.DvcCnfgRcrd_skipDigits);
                    if (bool_rc == FALSE)
                    {
                        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0,
NULL,
                            SOURCENAME, __LINE__, index, 0, NULL,
                            _T("bad add devices"));
                        rc = RR_MAPPING_FAILURE;
                        break;
                    }

                                        next_dev_p = next_dev_p-
>next;
                    index++;
                } // end while

                if (rc != GOOD)
                {
                                        ;
                }
                else if (acl_devices_list_rs_p->sttnCnfgBlckNmbr ==
0xFFFF)
                {

                    buffer_mgr& buffer_manager =
get_acl_buffer_mgr();
                    buff_rc =
buffer_manager.free_buffer(saved_data_p->buffer_p);
                    if (buff_rc != BUFF_MGR_RC_GOOD)
                    {
                        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0,
NULL,
                            SOURCENAME, __LINE__, 0, 0, NULL,
                            _T("can't free buffer"));
                    }

                                        acl_refid_mgr&
acl_refid_manager = get_acl_refid_mgr();
```

```
                    ref_rc = acl_refid_manager.free_user_request_id(
                        (USHORT) in_msg_p-
>u.acl_struct.acl_services.u.
                        returnResultsService.referenceNumber);
                    if (ref_rc != USER_REQID_MGR_RC_GOOD)
                    {
                        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0,
NULL,
                            SOURCENAME, __LINE__, 0, 0, NULL,
                            _T("can't free buffer"));
                    }

                    if (saved_data_p->device_request_type ==
START_UP_REQUEST)
                    {
                                                db_rc =
mp_db.create_devices(dev_list);
                        if (db_rc != MAPPER_DB_RC_GOOD)
                        {
                            su_log_message(0, SU_DEBUG,
SU_TRACE_NONE, 0, NULL,
                                SOURCENAME, __LINE__, db_rc, 0, NULL,
                                _T("bad database"));
                            rc = RR_MAPPING_FAILURE;
                        }
                    }
                    else if (saved_data_p->device_request_type ==
ADMIN_REQUEST)
                    {
                                                db_rc =
mp_db.update_devices(dev_list);
                        if (db_rc != MAPPER_DB_RC_GOOD)
                        {
                            su_log_message(0, SU_DEBUG,
SU_TRACE_NONE, 0, NULL,
                                SOURCENAME, __LINE__, db_rc, 0, NULL,
                                _T("bad database"));


                            db_rc = mp_db.create_devices(dev_list);
                            if (db_rc != MAPPER_DB_RC_GOOD)
                            {
                                su_log_message(0, SU_DEBUG,
SU_TRACE_NONE, 0, NULL,
                                    SOURCENAME, __LINE__, db_rc, 0,
NULL,
                                    _T("bad database"));
                                rc = RR_MAPPING_FAILURE;
                            }
                        }
                    }

                    cf_rc = cf_db.set_device_list(dev_list);
                    if (cf_rc != CNFG_DB_RC_GOOD)
                    {
                        su_log_message(0, SU_DEBUG, SU_TRACE_NONE,
                            saved_data_p->trace_correlator, NULL,
                            SOURCENAME, __LINE__, db_rc, 0, NULL,
                            _T("bad database"));
                        rc = RR_MAPPING_FAILURE;
                    }

                                        if (rc == GOOD)
                    {
                        pl_list_update_complete(TRUE);
                    }
```

```
                    else
                    {
                        pl_list_update_complete(FALSE);
                    }
                }
            }
        else
        {
            mp_refid_timeout()

            su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
                SOURCENAME, __LINE__, acl_devices_list_rs_p-
>acknowledgeCode, 0, NULL,
                _T("bad ackCode"));
            rc = GOOD;
        }
    }
        else
    {
        su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
            SOURCENAME, __LINE__, rc, sizeof(MP_STRUCT),

            (BYTE *) in_msg_p,
            _T("From get_acl_response_struct_pointer"));
        rc = RR_MAPPING_FAILURE;
    }
  }

  if (cf_db.debug_tracing_on(DBG_RQST_MAPPER_BIT) == TRUE)
  {
      wsprintf(trace_buff, _T("Exiting %s\n"), function_name);
      su_log_message(0, SU_DEBUG, SU_TRACE_NONE, 0, NULL,
          SOURCENAME, __LINE__, 0, 0, NULL, trace_buff);
  }

  return(rc);
}
```